Elizabethtown College

Raspberry Pi Lab #1 - **MOTION SENSOR**

by Daniel Esteves, 2017

1- THE RASPBERRY PI

As this is our first lab with the Raspberry Pi, I want to make sure you have everything to get started. The list of must-haves:

1. DVI to HDMI cable
2. MicroSD card with Raspbian Image loaded into it
3. Sensor kit
4. Mouse and keyboard
5. MicroUSB Power supply

If you don't have any of the above, please let the TA know.

2- THE SOFTWARE

The Raspberry Pi should be ready for use without any setups. The only thing you may need is to update the libraries and commands for terminal. It goes as follows:

- ***sudo apt-get update***
- ***sudo apt-get upgrade***

This shouldn't take much time, but expect it to be a time consuming process in the future, since more libraries, more time to check all.
For the beginning, we are going to use scripts in C and/or Python. If you are not used to it, don't worry. The goal is to make you dig and find solutions, not learn a new language.
This example I'm going to demonstrate uses Python.

2.1- THE EXAMPLE

Lets start by installing Python:

```
sudo apt-get install python-dev
sudo apt-get install python-pip
```

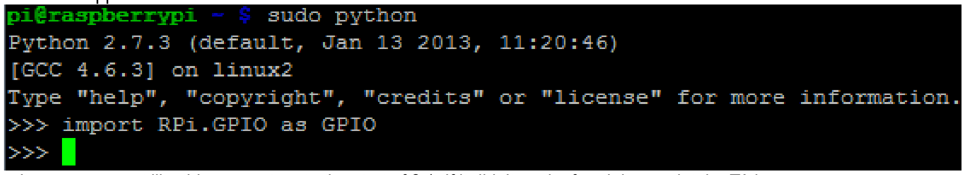
Now lets install **rpi.gpio**, a control module for GPIO channels

```
sudo pip install rpi.gpio
```

Test Python using:

```
sudo python
```

A prompt like this one should appear:



```
pi@raspberrypi ~ $ sudo python
Python 2.7.3 (default, Jan 13 2013, 11:20:46)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import RPi.GPIO as GPIO
>>>
```

If sudo python doesn't open a prompt like this one, return to the start of 2.1. If it didn't work after doing so, let the TA know.
Enter exit() to quit Python.

To create our first script? Enter the commands below:

```
sudo touch led.py
sudo nano led.py
```

If you're not familiar with Linux, nano is a text editor for Terminal. Sometimes you need to use it for permission problems and it provides a raw data file, different from programs like Word.

With **led.py** file open on terminal using nano, copy the following (This turns on and off an LED depending on motion sensor):

```
import RPi.GPIO as GPIO          #importing the RPi.GPIO module
import time                      #importing the time module
GPIO.cleanup()                   #to clean up at the end of your script
led_pin = 37                     #select the pin for the LED
motion_pin = 35                  #select the pin for the motion sensor
def init():
  GPIO.setmode(GPIO.BOARD)       #to specify which pin numbering system
  GPIO.setwarnings(False)
  GPIO.setup(led_pin,GPIO.OUT)                  #declare the led_pin as an output
  GPIO.setup(motion_pin,GPIO.IN,pull_up_down=GPIO.PUD_UP)  #declare the motion_pin as an input
  print("-----------------------------------------------------------------------")

def main():
  while True:
    value=GPIO.input(motion_pin)
    if value!=0:                 #to read the value of a GPIO pin
      GPIO.output(led_pin,GPIO.HIGH)      #turn on led
      time.sleep(2)    #delay 2ms
      print "LED on"               #print information
    else:
      GPIO.output(led_pin,GPIO.LOW)       #turn off led
      time.sleep(2)    #delay 2ms
      print "LED off"              #print information

init()
main()
GPIO.cleanup()
```

To Exit nano, press Ctrl+X, it will ask if you are sure, Press "Y" and enter for the **led.py** name.

Run the code by entering

```
sudo python ./led.py
```

This should load a screen, but wait... the pins are not connected... Lets fix this!

2.2 – THE HARDWARE
First of all, this exercise will be using the 5V port of the Raspberry Pi. Power the breadboard as you do with the circuit trainer. Connect the Vcc port of the sensor to 5V and ground, of course, to ground. The OUT port in the sensor needs to be connected to GPIO19. We connect an LED with the power coming from GPIO26 (don't forget the resistor).

2.3 – THE STUFF WORKING
Finally, we are ready to go back to the terminal window. You should rerun the command

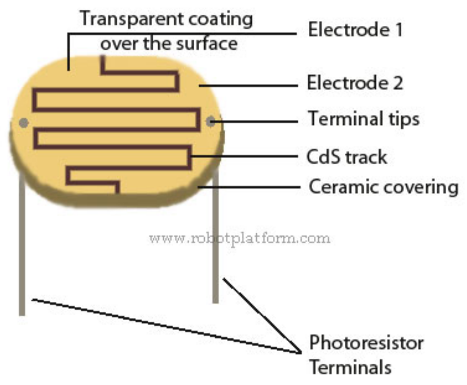
```
sudo python ./led.py
```

This will load a screen that will display LED ON/OFF depending if the sensor is triggering it.

3- NOW YOU!

(1) Check the **LED.PY** file. Try to understand exactly what each line does. For example, Why are led_pin= 37 and motion_pin= 35 if they are side-by-side?(**https://pinout.xyz/**).
(2) Using the same format, try to create a new project. You don't need to make anything too fancy, just use something from the sensor kit or something else from the lab. An example would be to make the photoresistor to trigger the LED, like a sustainable energy saving device. Of course, if you can integrate it to the Phoenix Contact NanoPLC, it is a plus. You may look online for tutorials for cool stuff to do, but, please, DO NOT just copy and paste the information. Try to understand exactly what you are doing.

Elizabethtown College
Raspberry Pi Lab #2 – **Anolog toDigital, and PHOTORESISTOR**
by Daniel Esteves 2017



1- INTRODUCTION
Also known as light-dependent resistor (LDR), the photoresistor adjusts its resistance according to the light received from the environment. It works not only with sunlight, but also with artificial light. Now lets see how we can integrate it to the real world.

2- THE RASPBERRY PI (RPI)
With your RPi turned on, make sure it is connected to the internet by going to the Internet Browser and trying to access google.com (don't try to access etown.edu since the college authorizes this communication even though you are not actually connected to the internet).
If you got the connection, open Terminal

3- THE TERMINAL

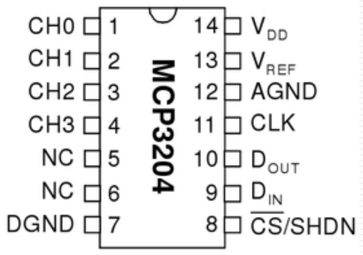
```
sudo apt-get update
sudo apt-get upgrade
```

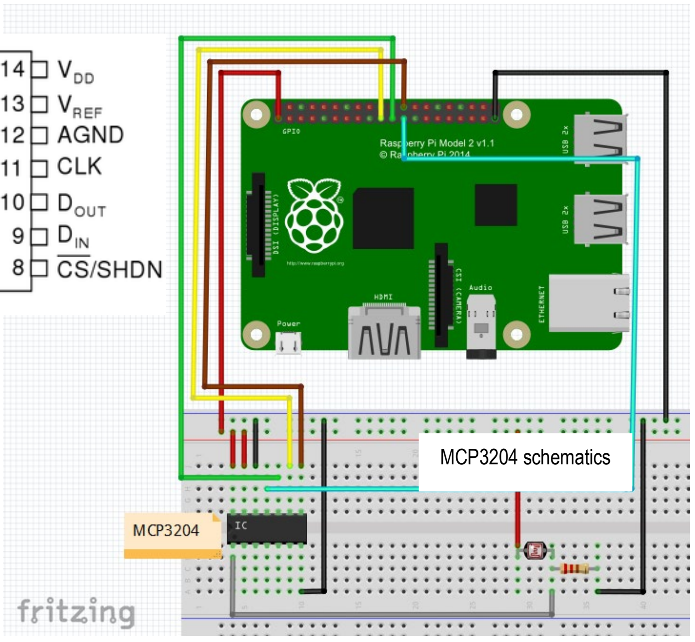
ALWAYS do this before messing with Terminal. It updates the libraries and commands.
This shouldn't take much time, but expect it to be a time consuming process in the future, since more libraries, more time to check all.
For this Lab, we will be using WiringPi to run a C script. It works pretty much like **rpi.gpio**, from the last lab, but uses C.

4.1- THE HARDWARE SETUP
You should connect the project as in the picture below:

This IC is an A/D Analog to Digital Convertor(MCP3204) and the **resistor is 10K Ohm**.

Use the website *https://pinout.xyz/ to find the pin names so you can connect to the right place.*





4.2- THE SOFTWARE SETUP
Lets start by installing GIT:

```
sudo apt-get install git-core
```

If you are wondering what is GIT, it is a version control system to track changes on your files and computer. Here is the link to the creators website: https://git-scm.com

After downloading GIT, you will be able to get WiringPi using this command:

```
git clone git://git.drogon.net/wiringPi
```

To build WiringPi

```
cd wiringPi
./build
```

Now lets create the Script by entering:

```
sudo touch res.c
sudo nano res.c
```

After you enter this command, nano will prompt so you can edit **res.c**. For this example, the code used can be seen below:

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include <unistd.h>
#include <wiringPi.h>
#include <wiringPiSPI.h>
#include <unistd.h>

#define CS_MCP3208  8       // BCM_GPIO8

#define SPI_CHANNEL 0
#define SPI_SPEED   100000  //


int read_mcp3208_adc(unsigned char adcChannel)
{
  unsigned char buff[3];
  int adcValue = 0;

  buff[0] = 0x06 | ((adcChannel & 0x07) >> 7);
  buff[1] = ((adcChannel & 0x07) << 6);
  buff[2] = 0x00;

  digitalWrite(CS_MCP3208, 0);  // Low : CS Active

  wiringPiSPIDataRW(SPI_CHANNEL, buff, 3);

  buff[1] = 0x0F & buff[1];
  adcValue = ( buff[1] << 8) | buff[2];

  digitalWrite(CS_MCP3208, 1);  // High : CS Inactive

  return adcValue;
}


int main (void)
{
  int adc1Channel = 0;
  int adc1Value   = 0;

  if(wiringPiSetup() == -1)
  {
    fprintf (stdout, "Unable to start wiringPi: %s\n", strerror(errno));
    return 1 ;
  }

  if(wiringPiSPISetup(SPI_CHANNEL, SPI_SPEED) == -1)
  {
    fprintf (stdout, "wiringPiSPISetup Failed: %s\n", strerror(errno));
    return 1 ;
  }

  pinMode(CS_MCP3208, OUTPUT);

  while(1)
  {
    system("clear");
    printf("\n\nMCP3208 channel output.\n\n");
    adc1Value = read_mcp3208_adc(adc1Channel);
    printf("adc0 Value = %04u", adc1Value);
    printf("\tVoltage = %.3f\n", ((3.3/4096) * adc1Value));
    usleep(1000000);
  }
  return 0;
}
```

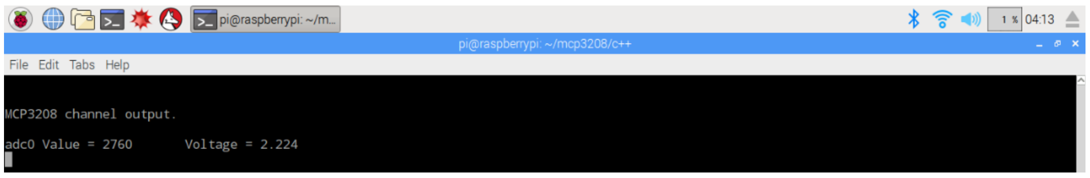
To Exit nano, press Ctrl+X, it will ask if you are sure, Press "Y" and Enter for the **res.c** name.
Compile the code by entering:

```
gcc -Wall -o app res.c -lwiringPi
```

Run the code with this command:

```
sudo ./app
```

It should display this screen:



```
MCP3208 channel output.

adc0 Value = 2760       Voltage = 2.224
```

5- NOW YOU!
Check the **res.c** file. Make the prompt, instead of displaying "MCP3208 channel output.", display "Lab 2 – Student name" on the prompt screen. Also, before each line of "adc0 Value = …" display the counting number. The output should be something like:

"Lab 2 – Daniel Esteves
1 – Voltage = 2.224"

Using the same **res.c** file, instead of showing values, trigger a LED to turn on if the voltage is lower or equal to 2V. This will require a little more work and you may want to check out this page(*https://projects.drogon.net/raspberry-pi/wiringpi/functions/*). You can also make something else, other than trigger a LED, but check out with the TA beforehand to see if it is ok.

Elizabethtown College
Raspberry Pi Lab #3 – **ARM ASSEMBLY CODE**
by Daniel Esteves 2017

1- INTRODUCTION
This lab is intended to introduce you to the ARM Architecture/Assembly Code. ARM has become the main processor for gadgets like Smart Phones, Tablets and the best one: Raspberry Pi. Lets go over some examples

2- EXAMPLES
This is the code for the "Hello World" file

```
    .data
string: .asciz "\nHello World!\n"
    .text
    .global main
    .extern printf
main:
    push {ip, lr}
    ldr r0, =string
    bl printf
    pop {ip, pc}
```

To assemble, link and run files on ARM assembly code we need Terminal.

3- TERMINAL
Lets start typing the traditional:

sudo apt-get update
sudo apt-get upgrade

You can use your favorite text editor in Terminal, but for this tutorial I will use nano.
Create the **assembly1.s** file by typing

sudo nano assembly1.s

With nano opened you can type the Hello World example from above.
Close the file by Pressing Control+X and when prompted if you would like to save the changes, press Y and save the file with the same name (assembly1.s).
Now that we have the file done, lets assemble it.

as –g –o assembly1.o assembly1.s

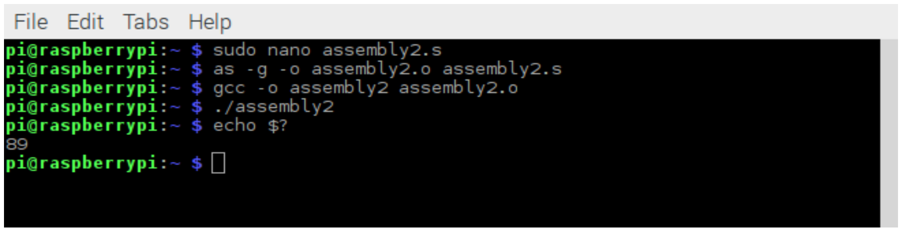
Now, link it.

gcc –o assembly1 assembly1.o

The file is now assembled and linked. You can now run it by entering

./assembly1

Your screen should look like this.


File Edit Tabs Help

```
pi@raspberrypi:~ $ sudo nano assembly2.s
pi@raspberrypi:~ $ as -g -o assembly2.o assembly2.s
pi@raspberrypi:~ $ gcc -o assembly2 assembly2.o
pi@raspberrypi:~ $ ./assembly2
pi@raspberrypi:~ $ echo $?
89
pi@raspberrypi:~ $ 
```

5- NOW YOU!
By doing the example you now know how to assemble an ARM assembly code program. Write a program that does the following:

(1) Move the integer 13 to the Register R1
(2) Move the integer 25 to the Register R2
(3) Put the sum of Register R1 and R2 to Register R3
(4) Move the value of R3 to R0

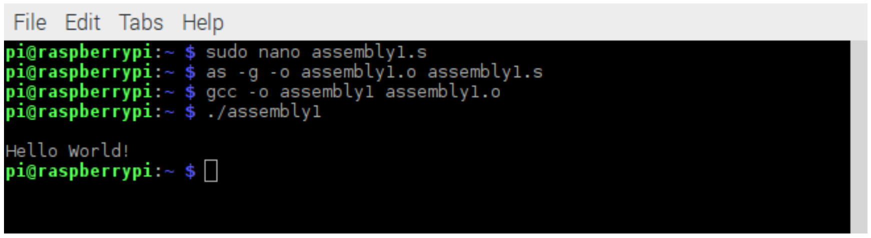
You will need to add the lines in the end of the file. This is called a system call and is needed to return the right value.

MOV R7, #1
SVC 0

If you did everything right, in terminal you can enter

echo $?

Giving the answer to the sum:


File Edit Tabs Help

```
pi@raspberrypi:~ $ sudo nano assembly1.s
pi@raspberrypi:~ $ as -g -o assembly1.o assembly1.s
pi@raspberrypi:~ $ gcc -o assembly1 assembly1.o
pi@raspberrypi:~ $ ./assembly1

Hello World!
pi@raspberrypi:~ $ 
```